SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Cmrk

IZRADA BEAT 'EM UP IGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Cmrk

Matični broj: 42839/14–R

Studij: Informacijski sustavi

IZRADA BEAT 'EM UP IGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor:

Dr. sc. Mladen Konecki

Varaždin, kolovoz 2018.

Matija Cmrk

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ovaj završni rad će obuhvaćati izradu računalne igre u razvojnom alatu i pokretaču Unity, odnosno izrade računalne igre žanra „Beat 'em Up“. Biti će ukratko rečeno što je to Unity i koje su njegove karakteristike, te što je to „Beat 'em Up“ žanr, njegovi predstavnici i mehanike. Zatim će biti opisan razvojni proces igre, stvaranja igrivog lika sa svim karakteristikama koje ulaze u to od dizajna i animacija do programiranja mehanika, također za neprijatelja, nivoa, sistema za nagrade, zvuka… U zaključku će biti izneseno moje iskustvo s radom u alatu, iskustvo samostalnog stvaranja igre u cjelokupnom procesu, te stajalište prema razvojnom alatu.

Ključne riječi: Unity; Beat em up; računalna igra; gamedev; programiranje; razvoj igre;

Sadržaj

1. Uvod

Moje djetinjstvo dobrim dijelom je obilježeno igranjem računalnih igara. Sjećam se kao da je bilo jučer kada sam proveo sate i sate igrajući Crash Bandicoot i slične igre na prvoj PlayStation konzoli. Industrija u to vrijeme nije bila velika, a moje znanje o njoj još manje. S vremenom počeo sam pratiti i industriju kao takvu ne samo zasebne igre i to me još više zainteresiralo. Pratio sam kako se industrija video igara razvijala iz „male“ sile koja je dolazila sa negativnim stigmama u globalnu velesilu koja je čak prerasla i filmsku industriju. Kako je rasla industrija tako je rasla i moja želja da se bavim razvojem video igara. To me inspiriralo u uzimanju izrade video igre kao temu ovog završnog rada, a Beat 'em Up žanr zbog toga što sam u vrijeme uzimanja teme igrao igru Castle Crashers, koja je istog žanra.

Odabir alata za izradu nije bio previše težak. Unity kao alat za razvijanje igara, poznat je po tome da je pristupačan i jednostavan za početi (u nekim situacijama i previše) pa snosi posljedicu negativnih stigmi jer se zbog njegove jednostavnosti na tržište „izbacuju“ gomila nedovršenih i loših igara, pa time dobiva stigmu da se u njemu ne mogu napraviti dobre i kvalitetne igre, što nije istina. Imamo mnogo primjera dobrih igara napravljenih u Unity-u npr. Cities Skylines koja nam pokazuje koliko zapravo Unity daje mogućnost razvijanja kvalitetne i kompleksne igre. I tako gledajući mane i prednosty Unity-a zaključio sam da bi taj alat bio savršen za izradu igre u žanru Beat 'em Up koju radim za ovaj završni rad.

2. Unity

Unity je pokretač (eng. game engine), a pokretač je softver koji omogućava kreatorima igara da svoje igre naprave brzo i efikasno. Pokretač je framework za razvoj igara koji spaja sve aspekte izrade igre u jedan (Unity technologies 2018).

Unity je izbačen na tržište u lipnju 2005. godine kao OS-X ekskluzivni pokretač. No danas se proširio na 27 drugih platforma, od PC-a, konzola, pa do pametnih ekrana (Wikipedia, 2018). To je uvelike pripomoglo mom odabiru Unity-a kao programski alat koji ću koristiti za izradu završnog rada.

Unity se može preuzeti u 3 verzije Personal, Plus i Pro. Personal verzija je besplatna i sadržava sve funkcije koje su potrebne za kreiranje igre, i namijenjen je za početnike koji se žele upoznati s Unityem (to je i verzija koju ću ja koristiti u ovom radu). Plus koja košta 25$ na mjesec uz 1 godinu plaćeno unaprijed ili 35$ na mjesec. Ta verzija sadrži Unity-ev savjetnik za uspjeh, kurseve za izradu igara, 20% manje sa njihovog dućana za digitalnu imovinu, pro izgled za Unity, naprednija analitika i koristi se ako se zarađuje ispod 200 000$ godišnje. Pro verzija ima sve kao i Plus verzija, dozvolu za pregled izvornog koda, te se koristi kada se zarađuje preko 200 000$ godišnje. (Unity Store, 2018)

Kada govorimo o mogućnostima koje nam Unity daje tu su mnoge. Kada želimo kreirati igru možemo odabrati da li ćemo raditi 2D ili 3D igru. Daje nam se program koji u sebi sadrži sve potrebne alate za izradu elemenata potrebnih za izradu igre, kao npr. za izradu animacija, pričanja priče, dizajniranje svijeta, uređivanje svijeta i mnogo drugih stvari koje izradu igre čine jednostavnijom. Tu uvelike i pomaže „Unity Asset Store“ koji je trgovno mjesto za nabavu besplatnih i plaćenih asseta direktno sa stranice koju pruža Unity, te daje mogućnost lagane ugradnje tih asseta u projekt (Unity technologies 2018).

Kao jezik programiranja u Unity-u se prvenstveno koristi C#, no može se koristiti i Boo za kojeg je podrška maknuta od verzije 5.0. i UnityScript (koji je zapravo prilagođeni JavaScript za Unity) (aleksandr 2014). Ja sam se u svome završnom radu odlučio koristiti C#.

Pošto Unity nudi gomilu mogućnosti i opcija koje možemo koristiti, na stranici Unity-a nalazi se kosristan i opširan priručnik pod imenom „Unity User Manual (2018.2)“ u kojem je dokumentirano sve što trebamo za rad u programu. Priručnik je podijeljen u više poglavlja, ovdje ću spomenuti i opisati najbitnije za ovaj rad. Rad u Unity-u, kako početi raditi i napredni rad. Poglavlje za uvoz koja govori o uvozu modela, postavki i slično. 2D poglavlje govori načinu igre u 2D-u, sprite-ovima, tilemapama i fiziki za 2D. Poglavlje za grafiku koje daje pregled svih grafičkih mogućnosti kao svijetlost, kamera, materijali i slično, te daje primjere na njih. 3D poglavlje daje pregled svojih opcija kao što su Rigidbody, Collideri… Poglavlje pod imenom

„Skriptiranje“ spominje sve od kako napraviti skriptu, do sitnih detalja kako programirati te koja pomagala Unity nudi. Poglavlje za zvuk nam govori o konceptu zvuka u Unity-u, i formate zvuka koje Unity podržava. Animacijsko poglavlje sadrži sve što je potrebno za izrade animacija u Unity-u, od kako kreirati animacijski klip, do kako koristiti animator i animacijski kontroler. Poglavlje UI nam govori o kanvasu, vizualnim komponentama, tekstu i kako kreirati dobar UI. Postoji još nekoliko poglavlja tipa Multyplayer i umrežavanje, Unity servisi i slično, no oni nisu potrebni za ovaj rad. (Unity User Manual (2018.2), 2018

3. Beat 'em Up

Igra je smještena u žanru Beat 'em Up. Inspiracija za ovaj žanr dolazi od igra tipa „Battle Toads“ (Wikipedia, 2018), „Castle Crashers“ (Castle Crashers web stranica, 2018), „Little Fighter 2“ (Little Fighter 2 Official Website, 2018) i ostale slične igre.

Cilj igre u Beat 'em Up igri je sa svojim likom prolaziti kroz nivo i kada se pojave neprijatelji njih kao što ime žanra govori „prebiti“, i tako kroz cijeli nivo dok se ne dođe do glavnog neprijatelja, te nakon što smo „prebili“ njega pobjeđujemo nivo.

Mehanike su sljedeće. Nivo na kojem se nalazi igrivi lik je najvećim dijelom uvijek ravna ploha te se proteže u desnom smjeru. Igrivi lik se po toj plohi može kretati u smjerovima gore, dolje, lijevo i desno (u nekim varijacijama igre može skakati). Kako se pojavljuju neprijatelji na nivou, igrač ih mora „prebiti“, a to se osigurava tako da kada igrač dođe do grupe neprijatelja, kamera će se zaustaviti i neće dozvoliti da igrač nastavi kroz nivo tako dugo dok ne „prebije“ sve neprijatelje koji su se pojavili. Da bi igrač „prebio“ neprijatelje igrivi lik ima nekoliko napada i kombinacije napada za odabrati ili u nekim verzijama specijalne napade ili oružja koja pronalazi na putu. Kada „prebije“ nekog neprijatelja, postoji mogućnost da neprijatelj baci neki predmet koji će igraču vratiti život ili će baciti zlatnike koji se kasnije za nešto koriste, ovisno o sistemu koji je implementiran u igri.

U ovom radu odlučio sam se zbog demonstriranja mehanika igre napraviti jedan nivo. Kao izbor igrač će moći izabrati između 3 igriva lika: Gambit, Spiderman i Wolverine. Svaki od njih će imati drugačije animacije te i time drugačiju duljinu napada, snagu napada, brzinu napada, brzinu kretanja, život i posebni napad. Na nivou će se pojaviti 5 različitih neprijatelja: Sentinel, Silver Samurai, Omega Red, Morrigan te Thanos koji je i glavni neprijatelj. Svi ti neprijatelji kao i igrivi likovi će imati različite animacije i statistike. Te će se na nivou pojaviti kutije koje će igrač moći razbiti te će kutije dati zlato ili eliksir koji će igraču vratiti život.

4. Građa igre

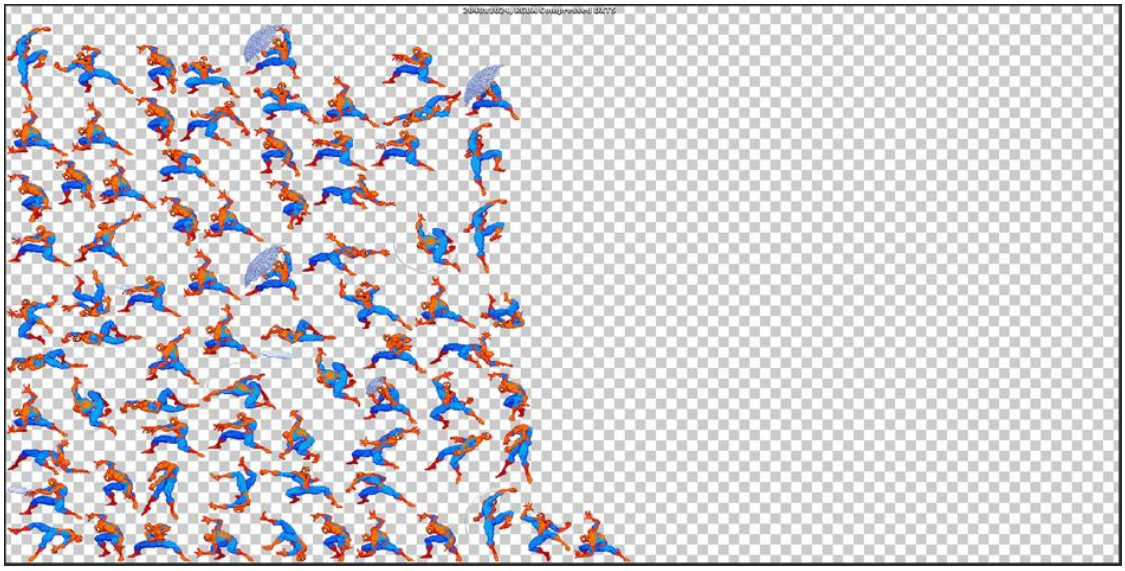
U ovom dijelu ću opisati kako sam izgradio sve elemente igre. Sve priložene slike biti će slikane u Unity verziji 2018.1.6f1.

4.1. Igrivi Lik

Kako je proces za stvaranje svih likova manje-više jednak opisivati ću izradu lika općenito a ne zasebno za svakog lika.

4.1.1. Stvaranje igrivih likova

Kako bi stvorio igrive likove prvo sam napravio novu scenu i na njoj podlogu na kojoj će lik stajati. Zatim sam unio sve slike tog lika te im promijenio Texture Type u „Sprite (2D and UI)“ i Packing Tag u ime lika. To sam napravio jer Unity ne može koristiti konvencijalne formate slika u svom radu, a Packing Tag sam postavio da si Unity može zapakirati sve slike lika na jedno mjesto za lakše dolaženje do njih. Ovdje ću staviti sliku zapakiranih slika za jednog lika a ostale možete vidjeti u prilogu.



Slika 1: Zapakirane sve slike za lika Spiderman (Izvor: Vlastita izrada)

Vrijedi spomenuti da su slike raspodijeljene u različitim mapama za svakog lika: Hodanje, Miran, Napad1, Napad2, Napad3, Obrana, Pad, Specijal i Udaren. Više o tome kasnije.

Da bi stvorio igrivog lika sada uzimam prvu sliku iz mape Miran i stavljam ju u scenu. Sada vidim lika u sceni, no to je samo slika, da bi postao pravi objekt u igri, moramo mu dodati komponente „Box Collider“ i „Rigidbody“. Te nakon toga postavljam Collider da lijepo obuhvaća mojeg lika u visini no jako malo po širini i dubini tako da kasnije kod nema problema sa smetanjem neprijateljima. Te pod rigidbody uključimo freeze rotation za x, y i z.

Sada imam objekt igrivog lika te mu dodajem tag „Igrac“ da bi mogao tražiti lika u kodu po tom tagu, ali to nije dovoljno, trebam mu dodati područje gdje može biti ranjen, na kojim mjestima on radi štetu, područje gdje ispucava specijalni napad i gdje će mu se neprijatelji pozicionirati.

Hitbox (mjesto gdje može biti ranjiv) stvaram tako da dodajem novi objekt kocke u scenu, postavim ga kao dijete igrivog lika. Box collider sada mora biti trigger i veličina se podesi tako da lijepo obuhvaća igrača u visini, širini i dubini. Također Mesh renderer mora biti isključen jer ne želimo vidjeti tu kocku.

Hitboxeve napada radim kao i hitbox lika samo što uzimam sliku u mapi napada u kojoj je najviše ispružen te prilagođavam box collider da obuhvaća to područje.

Područje ispucavanja specijala je prazni objekt koji postavimo kao dijete objekta na mjestu gdje želimo instancirati napad, ali o tome kasnije.

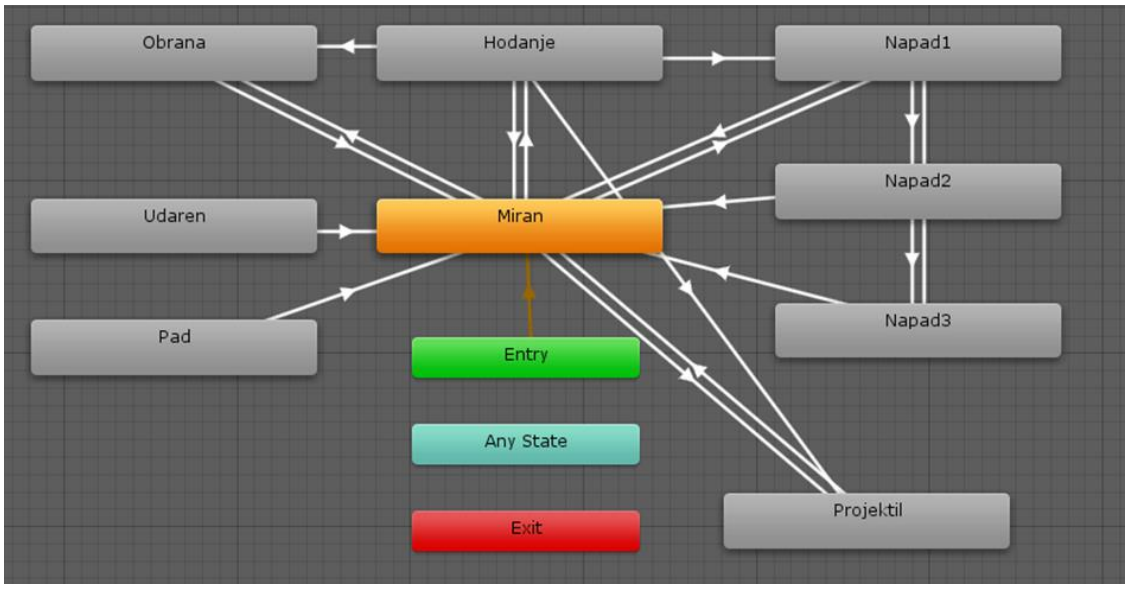
Te na kraju područje gdje će se neprijatelj pozicionirati su također prazni objekti, oni su pozicionirani malo ispred i malo iza našeg lika.

4.1.2. Animiranje igrivog lika

Unity za animiranje ima zgodnu stvarčicu koju zove „Animator“. Da bi koristio Animator trebam ga dodati na lika kao što sam ranije dodao Box Collider i Rigidbody. Nakon što smo ga dodali kreiramo datoteku animacijskog kontrolera i namjestimo ju u Animator. Sada otvaramo prozor „Animation“ i sada nastupaju ranije spomenute mape Hodanje, Miran… U svakoj mapi sam ranije razvrstao slike koje jedna za drugom čine prigodnu animaciju. Sada u Animation prozoru kreiramo novu datoteku za animaciju koju želimo pa iz mape samo povučemo u Animation prozor slike koje su povezane za animaciju i animacija je gotova, samo još trebamo posložiti koliko će uzoraka uzimati u sekundi tako da bi mogli izgladiti animaciju na brzinu koju želimo.

Nakon toga se animacije povezuju u prozoru „Animator“ tako da imamo svaku animaciju i povlačimo tranzicije između njih te uvijete kada želimo da se te animacije pokreću. Ti uvjeti su najčešće bool varijable no može biti nešto drugo npr float. Kasnije ću opisati kako aktiviram te animacije preko koda.

Još je važno napomenuti da animacije napada, specijal, pad i udaren moraju imati označeno da ima vrijeme izlaza jer želimo da se te animacije izvrše do kraja ako smo ih započeli. Također moramo animacijama za obranu i pad maknuti oznaku ponavljanja, jer ako pritisnemo i držimo tipku obrane ne želimo da on konstantno ulazi i izlazi iz poze obrane kao i što ne želimo kada je igrač oboren da se vrti animacije pada pa liči kao da pleše neki breakdance.



Slika 2: Animacijski kontroler za igrivog lika (Izvor: Vlastita izrada)

4.1.3. Kontroliranje igrivog lika

4.1.3.1. Kretanje

Kretanje je izvedeno tako da u skripti koja kontrolira lika u FixedUpdate metodi imamo 2 floata koja ispunjavamo s pozitivnim i negativnim brojevima ovisno kako se želimo kretati putem tipkovnice, tada te brojeve stavljamo u Vector3 varijablu preko koje možemo mijenjati poziciju našeg lika. Sada kada se lik kreće moramo uvesti i provjeru da li se krećemo lijevo ili desno pa ovisno o tome okrenuti lika. Te na posljetku treba proslijediti animatoru da počne izvoditi animaciju hodanja. To radim tako da sam u animatoru postavio da je uvjet float brzina te ako je brzina veća od 0.01 ulazim u animaciju a ako je manja izlazim. Pa tako samo preko koda svaki trenutak prosljeđujemo animatoru našu brzinu kretanja.

```
//--Kretanje-----------------------------------------------------------------------
float kretanjeHorizontalno = Input.GetAxis("Horizontal");
float kretanjeVertikalno = Input.GetAxis("Vertical");

Vector3 kretanje = new Vector3(kretanjeHorizontalno, 0.0f, kretanjeVertikalno);

rigidBody.velocity = kretanje * brzinaKretanja;
rigidBody.position = new Vector3
    (
        Mathf.Clamp(rigidBody.position.x, xMin, xMax),
        transform.position.y,
        Mathf.Clamp(rigidBody.position.z, zMin, zMax)
    );

if (kretanjeHorizontalno > 0 && !gledaDesno && moguKretati == true)
{
    Okreni();
}

else if (kretanjeHorizontalno < 0 && gledaDesno && moguKretati == true)
{
    Okreni();
}

animator.SetFloat("Brzina", rigidBody.velocity.sqrMagnitude);
```

Slika 3: Kod kretanja igrivog lika (Izvor: Vlastita izrada)

4.1.3.2. Napadanje

Za napadanje sam u animatoru stavio uvjet bool varijablu koja pokreće animaciju napadanja ako je istinita. Pa u FixedUpdate metodi provjerava ako je pritisnuta lijeva tipka miša, i ako je pritisnuta stavlja bool varijablu u istinu te ako nije na laž. Kao što smo spomenuo ranije, napravio sam hitbox-eve za svaki napad koji su triggeri, njih želim uključivati i isključivati ovisno u kojem smo dijelu animacije. To sam napravio tako da u svakom trenutku provjeravam koju sliku animacija prikazuje te ako je slika ista sa onom kojom želim da se izvrši napad upalim hitbox, a ako nije onda isključim.

```
//--Napadanje-----------------------------------------

if (Input.GetMouseButton(0))
    animator.SetBool("Napad", true);
else
    animator.SetBool("Napad", false);

if (napad1SpriteHitFrame == trenutniSprite.sprite)
    napad1Box.gameObject.SetActive(true);
else if (napad2SpriteHitFrame == trenutniSprite.sprite)
    napad2Box.gameObject.SetActive(true);
else if (napad3SpriteHitFrame == trenutniSprite.sprite)
    napad3Box.gameObject.SetActive(true);
else
{
    napad1Box.gameObject.SetActive(false);
    napad2Box.gameObject.SetActive(false);
    napad3Box.gameObject.SetActive(false);

}
```

Slika 4: Kod napadanja igrivog lika (Izvor: Vlastita izrada)

4.1.3.3. Obrana

Obrana je vrlo jednostavna. U animatoru imam bool varijablu koja pokreće animaciju te ako smo pritisnuli srednju tipku miša za obranu. Pali se varijabla u animatoru da se branimo i time pokreće animacija i još postavljamo bool u skripti da se branimo, ili u suprotnom se gasi.

```
//--Obrana--------------------------------

if (Input.GetMouseButton(2))
{
    animator.SetBool("Obrana", true);
    braniSe = true;
}
else
{
    animator.SetBool("Obrana", false);
    braniSe = false;
}
```

Slika 5: Kod obrane igrivog lika (Izvor: Vlastita izrada)

4.1.3.4. Udaren

Animacija udarenosti je izvedena tako da provjeravamo tri bool varijable, prije spomenutu da se branimo, da li smo oboreni i da li smo primili štetu. Ako smo primili štetu a ostale dvije su laž tada pokrećemo korutinu za primanje štete. U toj korutini se pali direktno animaciju da smo udareni, bez uvjeta, čeka da prođe vrijeme koje sam postavio da smo „ošamućeni“ od udarca, pa se nakon postavlja bool da primamo štetu na laž. Napomenuo bih kako ne moramo je gasiti animaciju jer animatoru nismo dali nikakav uvjet kada pali i gasi nego će se samo ugasiti kada završi.

```
//--Udaren--test-----------------------------------------------------

if (primioStetu == true && oboren == false && braniSe == false)
{
    StartCoroutine(PrimioStetu());
}
```

Slika 6: Kod udarenosti igrivog lika (Izvor: Vlastita izrada)

```
IEnumerator PrimioStetu()
{
    animator.Play("Udaren");
    moguKretati = false;

    yield return new WaitForSeconds(osamucenVrijeme);

    moguKretati = true;
    primioStetu = false;

}
```

Slika 7: Kod korutine udarenosti igrivog lika (Izvor: Vlastita izrada)

4.1.3.5. Oboren

Animacija oborenosti je slično izvedena kao i udaren, samo što se tu samo provjerava jedan bool da li je igrač oboren. Ako je oboren pokreće se korutina u kojoj pokrećemo animaciju pada ali postavljamo bool oboren u animatoru jer bi inače neovisno o vremenu čekanja korutine u animatoru mogla krenuti neka animacija nakon prvog ciklusa animacije pada. Pa sada nakon što prođe vrijeme koje smo postavili da smo oboreni, stavljamo bool u animatoru na laž i bool u skripti da smo oboreni na laž.

```
//--Oboren----------------------
if (oboren == true)
    StartCoroutine(Oboren());
```

Slika 8: Kod oborenosti igrivog lika (Izvor: Vlastita izrada)

```
IEnumerator Oboren()
{
    animator.Play("Pad");
    animator.SetBool("Oboren", true);
    moguKretati = false;

    yield return new WaitForSeconds(oborenVrijeme);

    animator.SetBool("Oboren", false);
    moguKretati = true;
    oboren = false;
}
```

Slika 9: Kod korutine oborenosti igrivog lika (Izvor: Vlastita izrada)

4.1.3.6. Specijal

Specijalni napad provjerava se na pritisak desne tipke miša. Ako je pritisnuta stavlja se bool u animatoru za specijalni napad na istinu te se pokreće animacija. Zatim se provjerava kao i u napadu da li je trenutna slika jednaka slici ispucavanja, i ako je instanciramo objekt specijalnog napada na mjesto ispucavanja. Te ako nije pritisnuta tipka bool u animatoru postavlja se na laž.

```
//--Specijal-----------------------------------------------------------------------

if (Input.GetMouseButton(1))
{
    animator.SetBool("Projektil", true);
    if (projektilSpriteThrowFrame == trenutniSprite.sprite)
    {

        if (gledaDesno)
        {
            GameObject noviSpecijal = Instantiate(specijalDesno, gameObject.transform.GetChild(6).position, gameObject.transform.GetChild(6).rotation);
            noviSpecijal.GetComponent<Rigidbody>().velocity = Vector3.right * brzinaSpecijala;
        }
        else
        {
            GameObject noviSpecijal = Instantiate(specijalLijevo, gameObject.transform.GetChild(6).position, gameObject.transform.GetChild(6).rotation);
            noviSpecijal.GetComponent<Rigidbody>().velocity = Vector3.right * -brzinaSpecijala;
        }
    }

}
else
{
    animator.SetBool("Projektil", false);
}
```

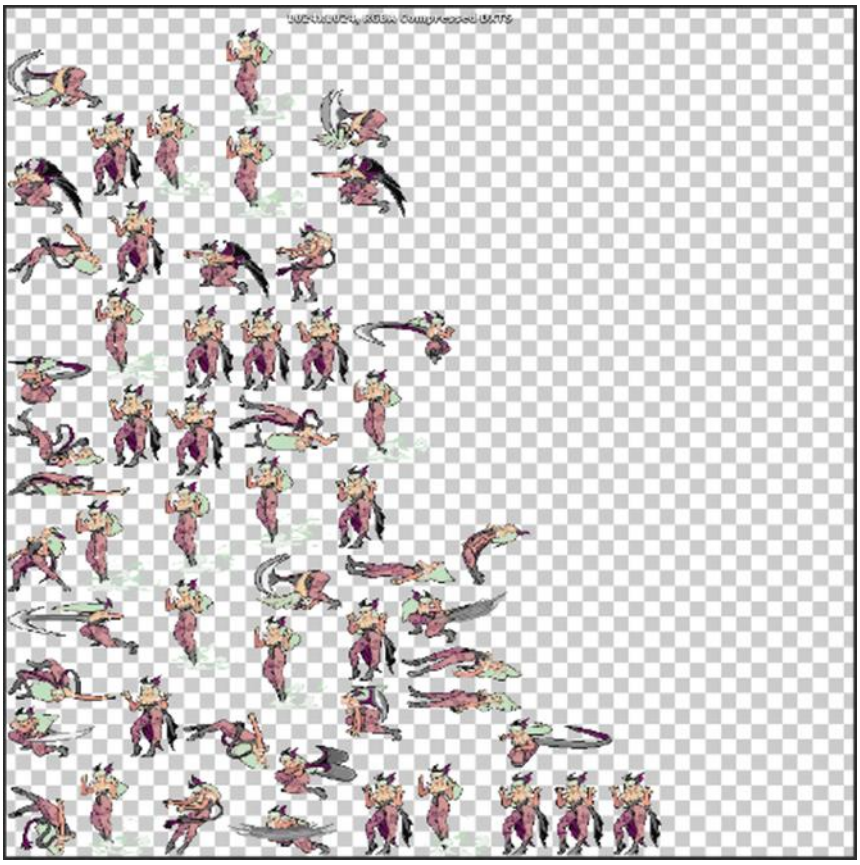
Slika 10: Kod specijala igrivog lika (Izvor: Vlastita izrada)

4.2. Neprijatelj

Izrada neprijatelja je vrlo slična izradi igrivog lika uz neke razlike, te ću i njih opisati.

4.2.1. Stvaranje neprijatelja

Proces stvaranja neprijatelja malo se razlikuje od procesa stvaranja igrivog lika. Na početku, isto kao i za stvaranje lika unosim slike u Unity, no neprijatelji nemaju animacije za spicijalni napad i obranu. Zatim ih također moram postaviti u „Sprite (2D and UI)“ i dodati im Packing Tag. Ovdje ću staviti zapakiranu sliku jednog neprijatelja a ostale možete vidjeti u prilogu.



Slika 11: Kod specijala igrivog lika (Izvor: Vlastita izrada)

Nakon što sam dodao slike, napravio sam isti proces kao i za igrivog lika, no ne stavljam mu Box Collider ni Rigidbody, nego samo Animator i izradim animacije. Radim novi prazni objekt te mu stavljam objekt s animatorom kao dijete, te mu postavljam Box Collider i Rigidbody. Sada mu dodajem Sphere Collider koji je trigger, i on će služiti kao vidno polje

neprijatelja. I na poslijetku kreiram Hitbox za neprijatelja i za njegove napade te mu ostavim tag „Neprijatelj“.

4.2.2. Umjetna inteligencija

4.2.2.1. Stanje neprijatelja

U ovom dijelu preko koda u skripti za stanje neprijatelja provjeravam je u kojem se stanju neprijatelj nalazi. Preko metode Update na početku provjeravamo je li život neprijatelja pozitivan i ako nije ga ubijamo, no više o tome kasnije. Zatim provjeravamo je li neprijatelj oboren kao kod igrivog lika, pa nakon toga je li udaren kao kod igrivog lika. Kod logike napada provjeravam da li je neprijatelj primio štetu što mora bit laž, i da li je igrač u vidokrugu što mora biti istina, i je li igrač oboren što mora biti laž, i je li napadačka daljina neprijatelja kraća od napadačke duljine i je li nav mesh agentova brzina kretanja manja od broja kojeg sam postavio kao početak napada. Tada u animatoru postavljam bool napada na istinu i bool kretanja na laž. Nakon toga provjeravam da li neprijatelj hoda prema igraču. Za to provjeravamo uvjete je li neprijatelj oboren oboren što mora biti laž, i jesmo li primili štetu što mara biti laž i je li igrač u vidokrugu što mora biti istina. Tada se postavlja obratno u animatoru nego što smo to radili za napad. I na posljetku provjeravam je li neprijatelj miran tako da bool za primanje štete i je li igrač u vidokrugu budu laž. Tada se u animatoru bool za kretanje i napadanje postavljaju na laž.

I na posljetku se provjerava da li je trenutno animacijsko stanje neprijatelja u mirnom stanju, hodanju ili napadanju, te se prema tome postavlja trenutno stanje neprijatelja preko enumeratora kojeg sam kreirao za ta 3 stanja.

4.2.2.2. Postavljanje prostora kretanja

Za ovaj dio koristio sam Unity-evu opciju za navigaciju „Nav Mesh Agent“. Dodamo Nav Mesh Agent kao komponentu na naš objekt neprijatelja. Postavim radius i visinu da lijepo okruži neprijatelja. Da bi Nav Mesh Agent radio treba otići u prozor „Navigation“ i pod opciju „Bake“ postaviti parametre i ispeći navigaciju za novo. To će uzeti u obzir objekte na nivou i napraviti polje kojim se neprijatelj može kretati.

4.2.2.3. Kretanje neprijatelja

Da bi se neprijatelj mogao kretati za početak mu treba napraviti detekciju igrača. To radim tako da radim skriptu za vid neprijatelja. U toj skripti koristim metode Unity-a „OnTriggerStay“ i „OnTriggerExit“ da bi postavio bool varijablu da li je igrač u vidokrugu na istinu ili laž. Također u metodi Update tražim relativnu poziciju igrača od neprijatelja te ako je

negativna postavljam bool varijablu koja provjerava je li igrač s desna na laž i obratno na istinu. Zatim provjeravam duljine prednjeg i stražnjeg cilja koje sam postavio na igraču, te ovisno je li prednji ili stražnji cilj bliže taj postavljam kao cilj prema kojem će se neprijatelj kretati.

Zatim kreiram skriptu za kretanje neprijatelja i u njoj u Update metodi provjeravam je li neprijatelj u stanju kretanja ili mirovanja, i ako se kreće pozivam funkciju za kretanje ili funkciju za stajanje. Funkcija za kretanje je napravljana tako da provjerava je li igrač s desna ili lijeva te okreće neprijatelja u tom smjeru. Provjerava je li neprijatelj oboren pa ako nije daje Nav Mesh Agentu postavljanu brzinu, te mu daje cilj koji smo postavili u skripti za vid. Dok funkcija za stajanje samo resetira Nav Mesh Agent.

```
// Update is called once per frame
void Update () {

    igracRelativnaPozicija = igrac.transform.position - gameObject.transform.position;

    if (igracRelativnaPozicija.x < 0)
        igracSDesna = false;
    else if (igracRelativnaPozicija.x > 0)
        igracSDesna = true;

    prednjiCiljDuljina = Vector3.Distance(prednjiCilj.transform.position, gameObject.transform.position);
    straznjiCiljDuljina = Vector3.Distance(straznjiCilj.transform.position, gameObject.transform.position);

    if (prednjiCiljDuljina < straznjiCiljDuljina)
        cilj = prednjiCilj;
    else if (prednjiCiljDuljina > straznjiCiljDuljina)
        cilj = straznjiCilj;

    daljinaNeprijatelja = Vector3.Distance(cilj.transform.position, gameObject.transform.position);
}

void OnTriggerStay(Collider other)
{
    if (other.gameObject == igrac)
        igracUVidokrugu = true;
}
```

Slika 12: Kod vida neprijatelja (Izvor: Vlastita izrada)

```
void Kretanje()
{
    if (vidNeprijatelja.igracSDesna == true && gledaDesno)
        Okreni();
    else if (vidNeprijatelja.igracSDesna == false && !gledaDesno)
        Okreni();

    if (stanjeNeprijatelja.oboren == true)
        navMeshAgent.speed = 0;
    else
        navMeshAgent.speed = brzina;

    trenutnaBrzina = navMeshAgent.velocity.sqrMagnitude;

    navMeshAgent.SetDestination(vidNeprijatelja.cilj.transform.position);
    navMeshAgent.updateRotation = false;
}
```

Slika 13: Kod kretanja neprijatelja (Izvor: Vlastita izrada)

4.2.2.4. Napadi neprijatelja

Napade neprijatelja postavljamo tako da radimo skriptu sa jednostavnim funkcijama za paljenje i gašenje napadačkih hitbox-eva. Zatim odlazimo u prozor „Animation“ gdje smo kreirali animacije i u animaciji na sliki kojoj želimo da se upali napad odaberemo funkciju za paljenje hitboxa i odmah sliku nakon nje odaberemo funkciju gašenje. Pošto nam stanje neprijatelja pali napade, sad još samo treba provjeriti je li neprijatelj u stanju napadanja i ako je treba resetirati Nav Mesh Agent da se neprijatelj ne kreće.

4.3. Kolizije

Kolizije su izvedene preko skripte za kolizije u kojoj preko metode OnTriggerEnter provjeravamo preko tagova čiji je napadački hitbox pogodio čiji hitbox. Ako je igrač pogodio neprijatelja poziva se funkcija koja provjerava je li neprijatelj oboren, i ako nije smanjujemo život neprijatelju i sviramo zvuk udarca. Ako je izvršen napad koji obara neprijatelja postavljamo mu bool varijablu za oborenost na istinu, ili ako nije postavljamo bool varijablu za primanje štete na istinu. Funkcija za primanje štete igrača je gotovo jednaka samo je razlika u tome što za sav ostali kod prvo provjeravamo da li je igrač u stanju obrane.

```
void OnTriggerEnter(Collider other)
{

    if (gameObject.tag == "IgracNapadBox" && other.tag == "NeprijateljHitBox")
    {
        NeprijateljPrimaStetu(other.gameObject);
    }
    else if (gameObject.tag == "NeprijateljNapadBox" && other.tag == "IgracHitBox")
    {
        IgracPrimaStetu(other.gameObject);
    }
    else return;
}
```

Slika 14: Kod provjere kolizije (Izvor: Vlastita izrada)

```
void NeprijateljPrimaStetu(GameObject other)
{
    drugiObjekt = other.transform.parent.gameObject;
    stanjeNeprijatelja = drugiObjekt.GetComponent<StanjeNeprijatelja>();
    statistike = drugiObjekt.GetComponent<Statistike>();

    if (stanjeNeprijatelja.oboren == false)
    {
        statistike.zivot -= snagaNapada;
        FindObjectOfType<AudioMenager>().Sviraj("Udarac");
    }


    if (oborenNapad == true)
        stanjeNeprijatelja.oboren = true;
    else
        stanjeNeprijatelja.primioStetu = true;
    Debug.Log("Neprijatelj je pogoden");
}
```

Slika 15: Kod za primanje štete neprijatelja (Izvor: Vlastita izrada)

```
void IgracPrimaStetu(GameObject other)
{
    drugiObjekt = other.transform.parent.gameObject;
    stanjeIgraca = drugiObjekt.GetComponent<IgracKontrolerSkripta>();
    statistike = drugiObjekt.GetComponent<Statistike>();

    if(stanjeIgraca.braniSe == false)
    {
        statistike.zivot -= snagaNapada;
        FindObjectOfType<AudioMenager>().Sviraj("Udarac");

        if (oborenNapad == true)
            stanjeIgraca.oboren = true;
        else
            stanjeIgraca.primioStetu = true;

        Debug.Log("Igrac je pogoden");
    }

}
```

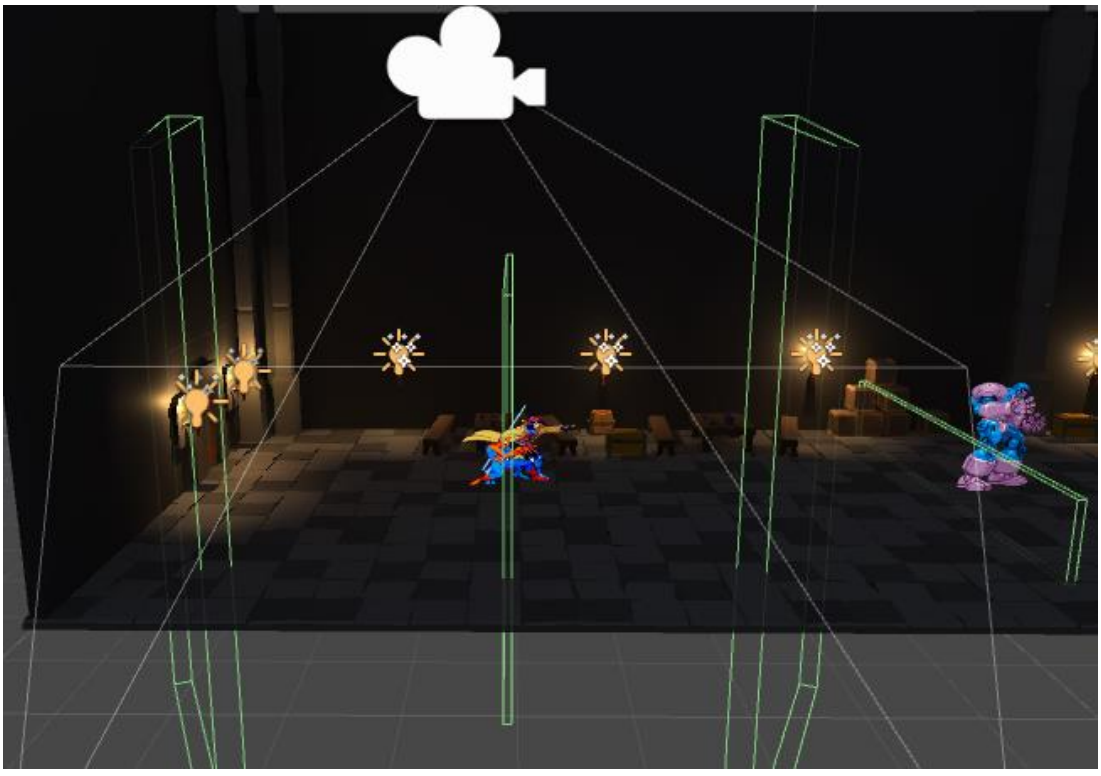
Slika 16: Kod za primanje štete igrača (Izvor: Vlastita izrada)

4.4. Kamera

Kamera ima vrlo bitnu ulogu u ovoj igri. Kao što je najčešća praksa u Beat 'em Up igrama želimo da nas kamera prati, te kada dođemo do odrečenog dijela mape želimo da se zaustavi te da ne možemo napredovati tako dugo dok ne pobijedimo sve neprijatelje na mapi.

To realiziram tako da kreiram prazan objekt pod nazivom Kamera Kontroler i stavim mu kameru kao dijete, zatim kreiram 2 objekta kocke kao djecu i razvučem ih do tako da one mogu obuhvatiti cijeli nivo po dubini. Te ih pozicioniram tako da se nalaze točno vidokruga mape i mijenjam im svojstvo da jedino s čime mogu imati koliziju je igrač. To će spriječiti igrača da izađe iz vidnog polja kada se kamera zaustavi. Zatim kreiram još jednu kocku kao dijete i stavljam ju na sredinu vidnog polja i postavljam da je trigger i isključujem mesh renderer jer ju ne želim vidjeti na ekranu, ona će služiti za okidanje sfera koje će zaustaviti kameru kada ih pogodimo.

Još jednu funkcionalnost koju želim da kamera kontroler ima je da na određenim mjestima okida stvaranje neprijatelje, to radim tako da mu dodam još jednu kocku kao dijete postavim ju da je trigger i pozicioniram tamo gdje završava prednji zid kamere i razvučem ga da pokriva cijelu dubinu nivoa. To će osigurati da se neprijatelji stvore točno prije nego bi se pojavili u vidnom polju.



Slika 17: Kamera kontroler (Izvor: Vlastita izrada)

U kodu sam to realiziramo tako da uvijek provjeravamo je li broj neprijatelja veći od nule. Ako nije veći od nule postavlja se bool za praćenje na istinu i kamera će nas početi pratiti. Kada zid za okidanje neprijatelja pogodi sferu koja će stvoriti neprijatelja te taj broj postaje veći od nule. No kamera se neće zaustaviti odmah nego tek kada središnji zid pogodi sferu za zaustavljanje kamere i to će prebaciti bool na laž i kamera će se zaustaviti. Sada moramo pobijediti sve neprijatelje i kada ih pobijedimo konstantnim provjeravanjem će vidjeti da je broj neprijatelja nula postaviti bool na istinu i početi će pratiti.

4.5. Životna traka

4.5.1. Dizajn

Da bi napravio životnu traku trebam kreirati novi UI element Canvas. Zatim mu kreiram kao dijeti prazni objekt koji će biti životna traka igrača, te njemu kreiram kao dijete objekt Slider. Sada označim dijete Slidera pod imenu Backround i mijenjam mu sliku na dizajn koji sam kreirao.

Slika 18: Oblik životne trake (Izvor: Vlastita izrada)

Zatim označujem Fill Area i njegovo dijete Fill i mijenjam sliku u zelenu traku koju sam napravio. Te ju pozicioniram na životnu traku u prazan prostor. I u hijerarhiji mičem iznad Backround tako da kod prikaza bude iza njega pa time bolje izgleda.

Slika 19: Prikaz života u životnoj traci (Izvor: Vlastita izrada)

Zatim pod Slider kreiram kao dijete novi sliku u koju stavljam crvenu podlogu za estetiku i pozicioniram ju iznad Fill Area tako da bude zadnja prikazana.

Slika 20: Crvena podloga životne trake (Izvor: Vlastita izrada)

I na kraju dodaje pod objektom životne trake kreiram sliku u koju ču stavljati portret igrivog lika ili neprijatelja i pozicioniram ga na sredinu kruga životne trake.

Time je završilo dizajniranje životne trake igrača te za neprijatelja samo kopiram traku i joj postavim veličinu x na -1 da dobijem zrcalnu sliku i pozicioniram ju na drugi kraj ekrana.

4.5.2. Implementacija

U skripti statistike gdje je spremljen život igraća kod pokretanja igre stavimo da je varijabla život ima vrijednost početnog života lika i isključimo sliku na životnoj traci.

U metodi Update postavljamo Slider-ovu maksimalnu vrijednost na vrijednost početnog života, zato ako udarimo neprijatelja s različitim životom da namjesti maksimalnu vrijednost Slidera prema tom naprijatelju. Zatim stavljamo sliku na životnoj traci i namještamo ju na sliku tog lika za kojeg gledamo. I na posljetku vrijednost Slidera postavljamo na vrijednost trenutnog života.

4.6. Zvuk

Zvuk sam u igru implementirao tako da sam napravio menadžer zvuka koji pokreće potreban zvuk kada mi treba.

Da bi implementirao menadžer prvo trebam napraviti klasu Zvuk u kojoj imam pojedinosti o zvuku, string ime, AudioClip clip (za datoteku zvuka koju želim), bool ponavljaj (ako želim da se ponavlja), float jacina i visina, i AudioSource izvor (za izvor zvuka u igri).

Nakon klase Zvuk trebamo napraviti skriptu koja će ju koristiti. U njoj napravimo listu klase Zvuk, te kod pokretanja nivoa sve podatke o njima stavimo izvor zvuka i funkciju koja će svirati te zvukove. I pokrećemo temu ovisno o liku kojeg smo izabrali.

Sada kada imamo skriptu možemo napraviti prazan objekt u koji dodamo tu skriptu. Sada u taj objekt možemo dodavati zvukove i podesiti ih, i taj objekt će proizvoditi potreban zvuk u igri.

Sada kada imamo menadžer zvuka, da bi bilo gdje u kodu koristili menadžer zvuka trebamo samo unijeti liniju koda FindObjectOfType<AudioMenager>().Sviraj("IME_ZVUKA");

4.7. Plijen

U igri sam implementirao sustav plijena koji se može dobiti kada se unište kutije koje se nalaze na nivou ili kad se ubije protivnik. Plijen može biti zlato ili eliksir koji vraća život, oboje imaju nasumičnu šansu stvaranja s prednošću na zlato.

Kada pokupimo zlato generira se nasumični broj zlata koji nam poveća prikazani broj zlata, a kada pokupimo eliksir on nam vrati 20 životnih vrijednosti.

4.8. Dizajn scena/nivoa

4.8.1. Glavni izbornik (scena 1)

Za izradu glavnog izbornika napravio sam praznu scenu i u nju stavio Canvas. Dodao sam crnu pozadinu te dodao tekst naslov igre, napravio sam prazni element koji sam nazvao početni izbornik i u njega stavio dva dugmeta IGRAJ i IZLAZ, zatim sam napravio novi prazni element u koji sam dodao tekst i tri dugmeta za svakog od igrivih likova, nazvao ga sporedni izbornik te ga stavio da nije aktivan. Za oblikovanje teksta koristio sam asset iz Unity Store-a Text Mesh Pro

Dugme IGRAJ ima funkciju da trenutni meni sa dva dugmeta isključi i uključi sporedni meni. Dugme IZLAZ ima funkciju da ugasi program.

Sva tri dugmeta u sporednom izborniku imaju funkciju da pokrenu scenu nivoa i spreme u statičnu klasu broj lika kojeg smo odabrali.

Slika 21: Početni izbornik (Izvor: Vlastita izrada)

Slika 22: Sporedni izbornik (Izvor: Vlastita izrada)

4.8.2. Nivo (scena 2)

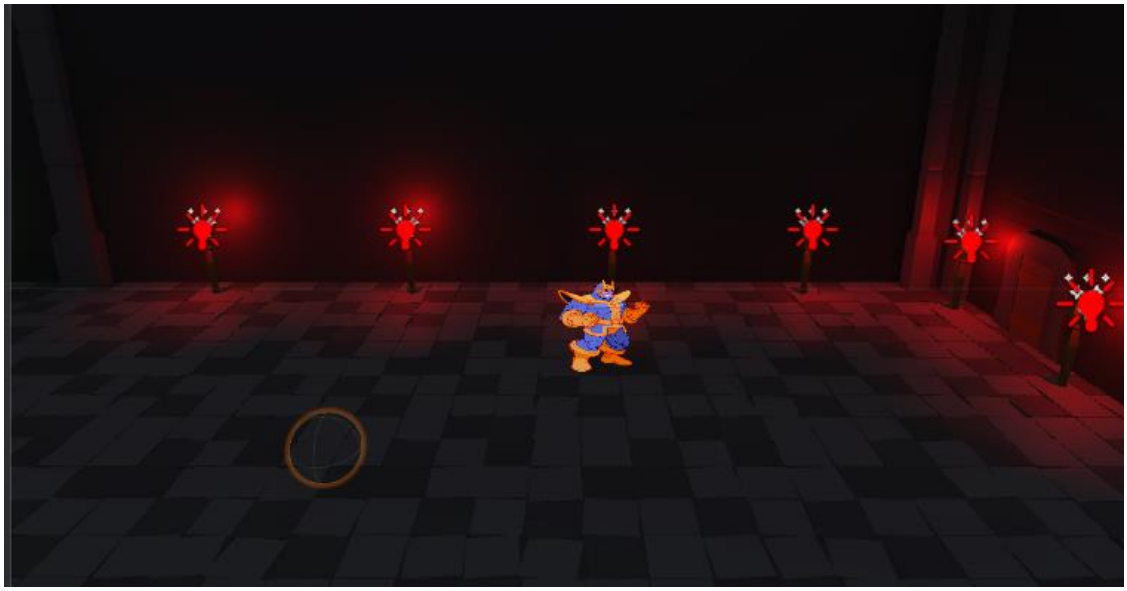
Kod dizajna nivoa koristio sam asset iz Unity Store-a pod nazivom LowPolyDungeonAssets. Ta skupina asseta je imala sve da napravim nivo kakav sam htio.

Ispod nivoa pozicionirao sam 4 sfere koje zaustavljaju kameru, i po nivou 7 stvarača neprijatelja. Na početku nivoa nalaze se sva 3 igriva lika na jednom mjestu pa ovisno o izboru iz prošle scene koji je spremljen u statičnoj klasi odabira lika ugasimo ostala 2 koja nisu izabrana.

Slika 23: Nivo s prvom sferom za zaustavljanje kamere (Izvor: Vlastita izrada)

Slika 24: Nivo s drugom sferom za zaustavljanje kamere (Izvor: Vlastita izrada)

Slika 25: Nivo s trećom sferom za zaustavljanje kamere (Izvor: Vlastita izrada)



Slika 26: Nivo s četvrtom sferom za zaustavljanje kamere (Izvor: Vlastita izrada)

4.8.3. Pobjetnički prikaz (scena 3)

Cilj pobjede u ovoj igri je ubiti zadnjeg neprijatelja koji je i pobjednički neprijatelj, i kada to napravimo prebacujemo se iz scene nivoa u scenu pobjedničkim prikazom.

Pobjednički prikaz napravljen je kao i početni izbornik sa 2 dugmeta ali bez sporednog izbornika. Dugme IZLAZ zatvara program, a dugme IGRAJ OPET pokreće početni izbornik.

Slika 27: Scena Pobjednički prikaz (Izvor: Vlastita izrada)

4.8.4. Gubitnički prikaz (scena 4)

Kada igramo igru, postoji rizik da nas neprijatelji mogu udarati, ako nas previše udaraju, ostajemo bez života, te kada život padne na nulu, izgubili smo. Tada se poziva scena Gubitnički prikaz koja je jednaka kao i Pobjednički prikaz u funkciji samo ima drugačiju poruku.

Slika 28: Scena Gubitnički prikaz (Izvor: Vlastita izrada)

5. Zaključak

Kada sam krenuo raditi igru, nakon što sam proučio materijale isplanirao sam da bi proces izrade mogao trajati od 20 do 30 sati. No tu sam se prevario, ispalo je gotovo duplo. Nisam podcijenio količinu posla već širinu posla. Od kodiranja do svakojakih dizajna, teško je izjednačiti tempo svih tih dijelova te tranzicijom između njih također stradava tempo rada. To mi je proširilo pogled na zašto postoje takvi ogromni timovi za izradu video igra.

Ovaj rad je bio prvi put da sam malo ozbiljnije radio u Unityu i išao u neke dublje dijelove njega. Naravno Unity ima još više elemenata koje nisam ni taknuo te bi ih trebalo istražiti detaljnije, no za ovaj rad ovo što sam koristio bilo je dovoljno. Taj dio koji sam koristio mi se jako svidio. Unity je mnogo olakšao proces izrade video igre u usporedbi sa nekim drugim alatima koje sam isprobao te je to jako veliki plus. Imam jednu zamjerku na „bug“ koji se dogodio u radu, a to je da se u jednom dijelu kad sam u Animatoru radio u sceni nivoa na tranziciji između animacija Nav Mesh Agent se počeo ponašati čudno i nije htio raditi kako treba, kada bih kliknuo na tranziciju animacije te nakon toga pokrenuo igru da testiram da li radi kako treba neprijatelj bi počeo hodati prema zidu a ne prema moje igraču. Potrošio sam par sati na pokušavanje otklanjanja buga, no bez uspjeha, jedino rješenje je bilo kod rada s Animatorom otvoriti novu scenu koja nema Nav Mesh Agent. No vjerujem da je to izoliran slučaj, tako da ne šteti mojem iskustvu s alatom, te bih ga preporučio svakome tko se želi iskusiti u izradi video igra.

Popis literature

[1] Unity Technologies *Game engines—how do they work?*
Preuzeto 14. Srpnja 2018 s https://unity3d.com/what-is-a-game-engine

[2] Unity Technologies *The world's leading content-creation engine*
Preuzeto 14. Srpnja 2018 s https://unity3d.com/unity

[3] Unity Technologies (2018, 9. srpnja) *Unity User Manual (2018.2)*
Preuzeto 14. Srpnja 2018 s https://docs.unity3d.com/Manual/

[4] Wikipedia, The Free Encyclopedia - *Unity (game engine)*
Preuzeto 17.7.2018 s https://en.wikipedia.org/wiki/Unity_(game_engine)

[5] Wikipedia, The Free Encyclopedia - Battletoads (video game)
Preuzeto 19.8.2018 s https://en.wikipedia.org/wiki/Battletoads_(video_game)

[6] Castle Crashers - web stranica
Preuzeto 19.8.2018 s https://www.castlecrashers.com/

[7] Little Fighter 2 Official Website
Preuzeto 19.8.2018 s http://lf2.net/

[8] Unity Store – web stranica
Preuzeto 25.8.2018 s https://store.unity.com/

[9] aleksandr–Unity Blog (2014, 3 rujan) *Documentation, Unity scripting langauages and you*
Preuzeto 18.7.2018 s https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/

[10] Patrck Reoder (2015) *SGD 213 Programing II*
Preuzeto 15.7.2018 s https://www.youtube.com/watch?v=Q9A_WbIeXIQ&list=PLdAZlT2Z_lOv3jQXAjXNqZXsp3Tf5Cjhk

[11] Brackeys (2017, 31 svibnja) *Introduction to AUDIO in Unity*
Preuzeto 15.7.2018 s https://www.youtube.com/watch?v=6OT43pvUyfY

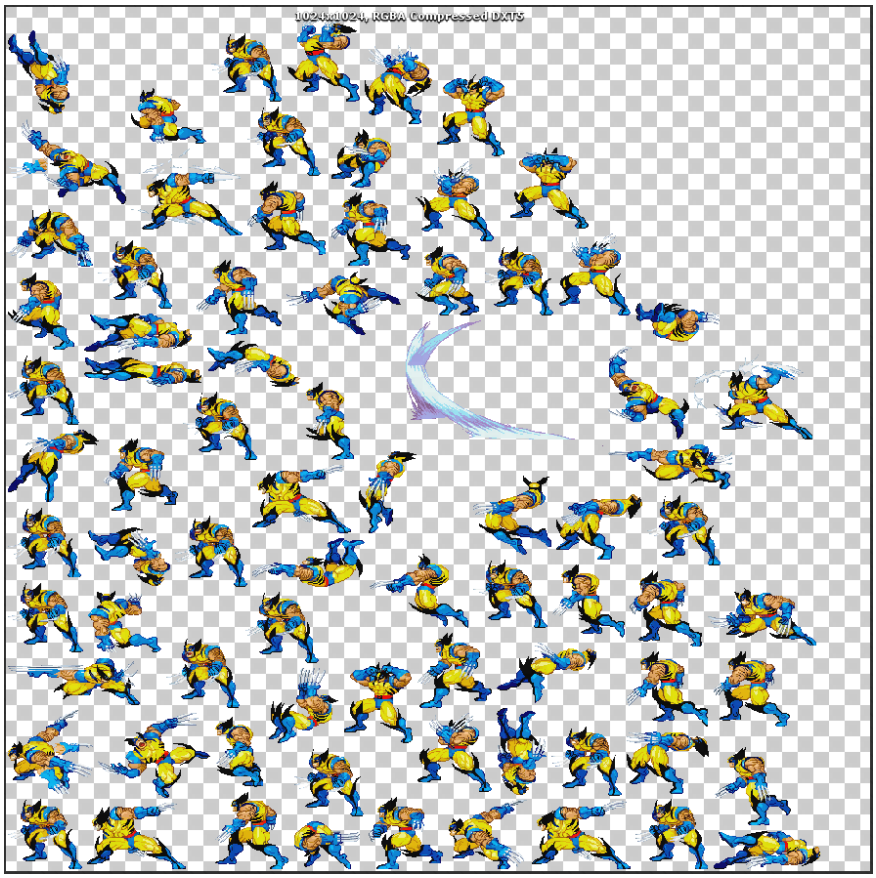
Popis slika

Popis korištenih resursa

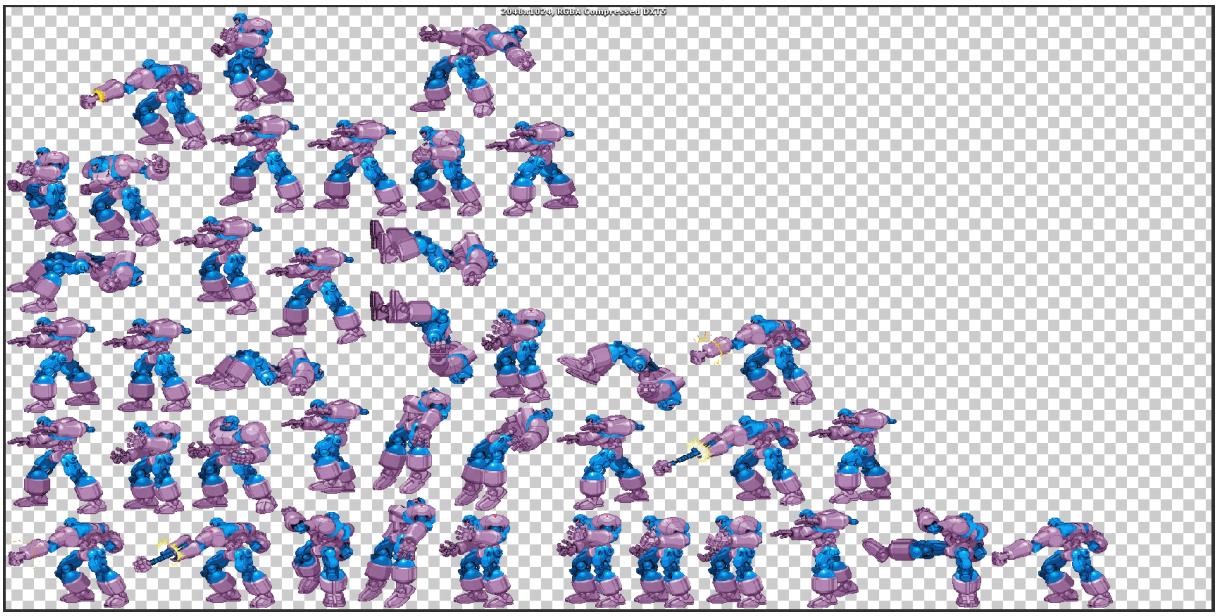
1. Slike svih likova - http://mugenguild.com/forum/topics/marvel-vs-capcom-2-sprites-157823.0.html dostupno 15.7.2018
2. Lowpoly Dungeon Assets (asseti za dizajn nivoa) - https://assetstore.unity.com/packages/3d/environments/dungeons/lowpoly-dungeon-assets-117330 dostupno 15.7.2018
3. TextMesh Pro (asset za oblikovanje teksta) - https://assetstore.unity.com/packages/essentials/beta-projects/textmesh-pro-84126 dostupno 15.7.2018
4. Zvuk udaranja - https://freesound.org/people/Ekokubza123/sounds/104183/ dostupno 15.7.2018
5. Zvuk za lik Gambit - https://www.youtube.com/watch?v=lwk_8sRlJPY dostupno 15.7.2018
6. Zvuk za lik Spiderman - https://www.youtube.com/watch?v=LNQi8xg-xek dostupno 15.7.2018
7. Zvuk za lik Wolverine - https://www.youtube.com/watch?v=kQ06visEn_o dostupno 15.7.2018

Prilozi

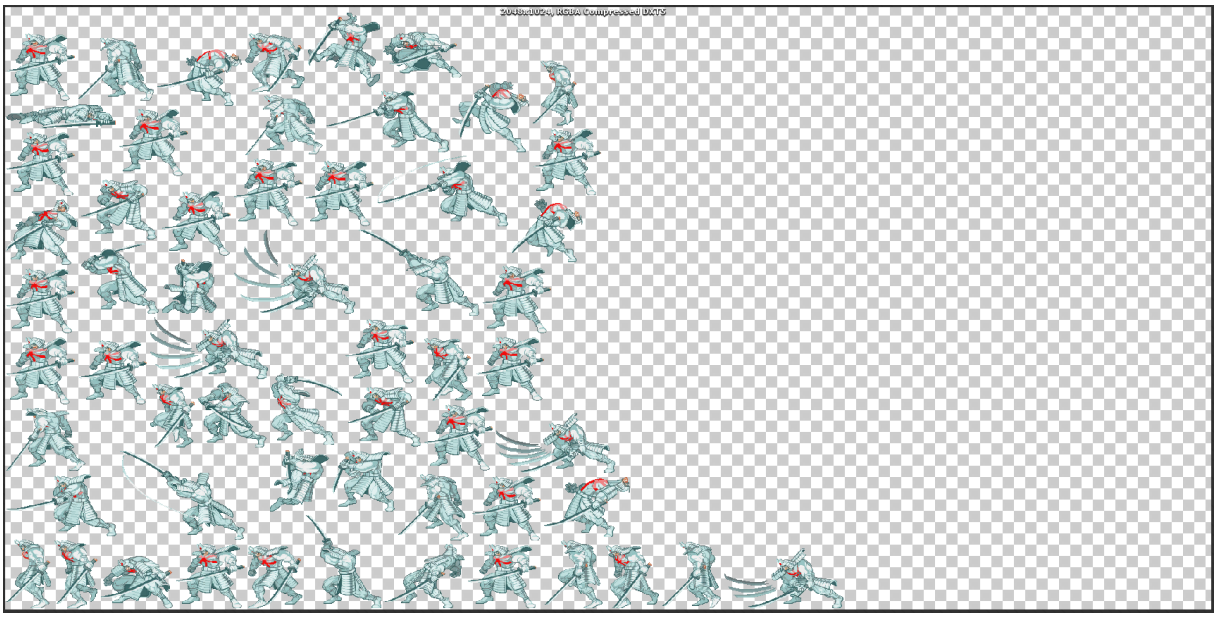
Slika 29: Zapakirane sve slike za lika Gambit (Izvor: Vlastita izrada)



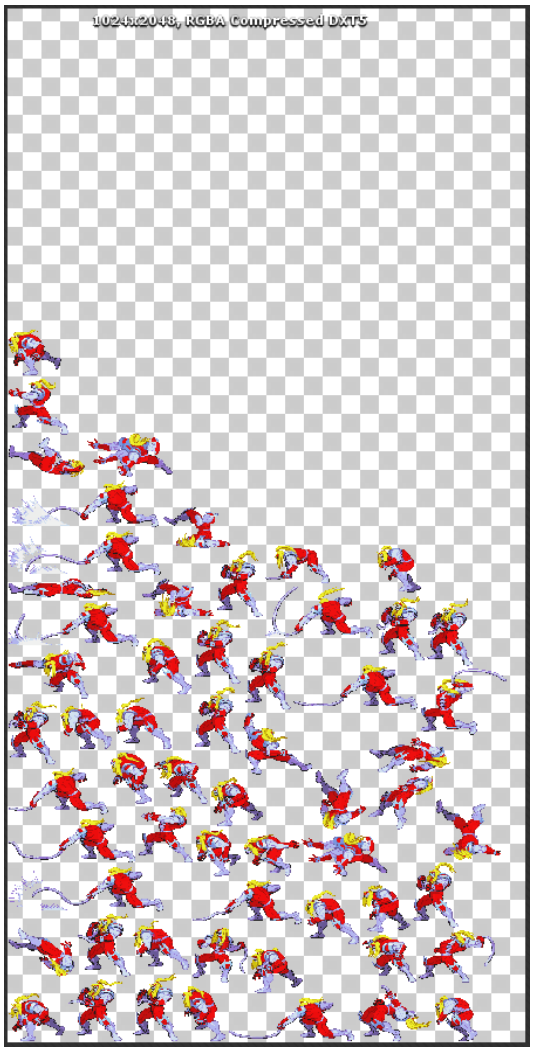
Slika 30: Zapakirane sve slike za lika Wolverine (Izvor: Vlastita izrada)



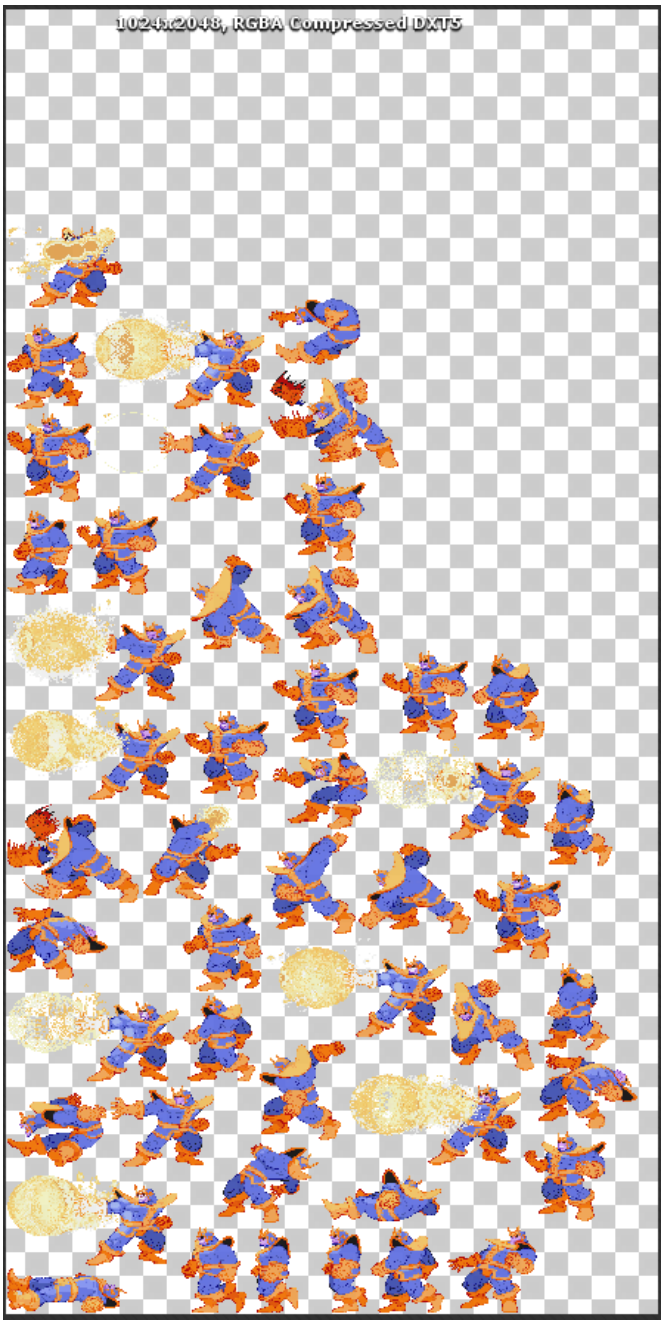
Slika 31: Zapakirane sve slike za lika Sentinel (Izvor: Vlastita izrada)



Slika 32: Zapakirane sve slike za lika Silver Samurai (Izvor: Vlastita izrada)



Slika 33: Zapakirane sve slike za lika Omega Red (Izvor: Vlastita izrada)



Slika 34: Zapakirane sve slike za lika Thanos (Izvor: Vlastita izrada)